Designing a Scripting Interface

Session 103

These are confidential sessions—please refrain from filming or taking pictures.

Application Technologies

Designing a Scripting Interface

Christopher Nebel
AppleScript and Automator Engineering

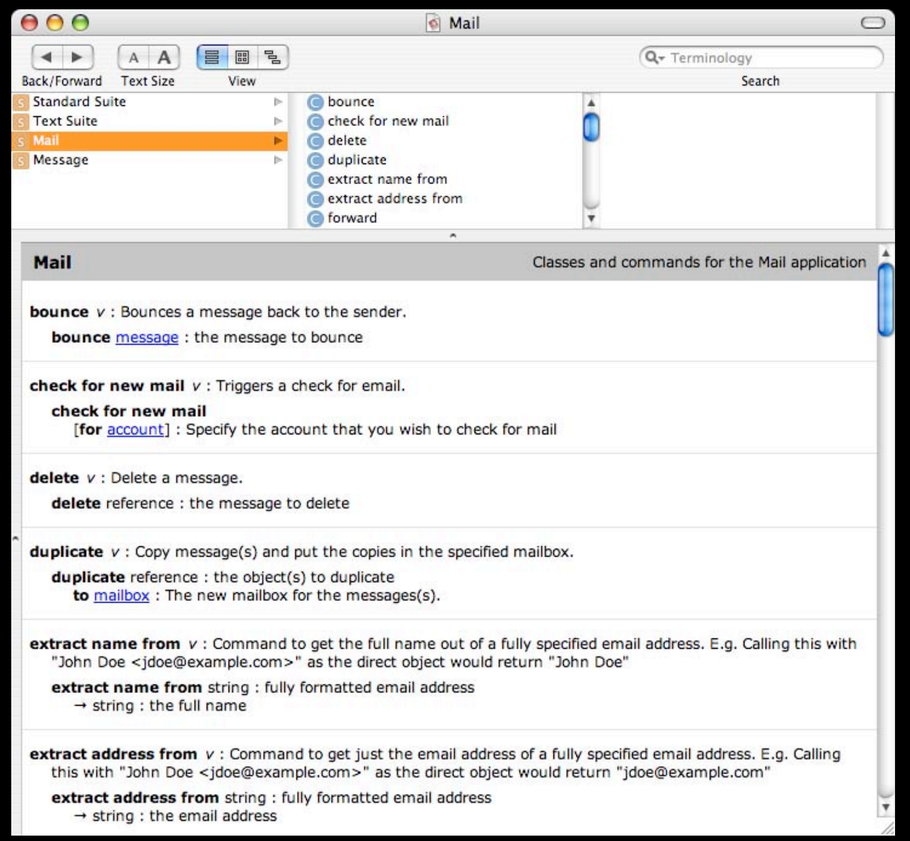
http://developer.apple.com/wwdc2005/

Introduction

Designing a Scripting Interface	Session 103 Tuesday 2:00PM	Hands On
Implementing a Scripting Interface	Session 112 Wednesday 9:00AM	Hands On
Lab: Make Your Application Scriptable	Session 1009 Wednesday 2:00PM	

What You'll Learn

- What a scripting interface is
- What it should look like
- Hands on: How to write an sdef (scripting definition) file

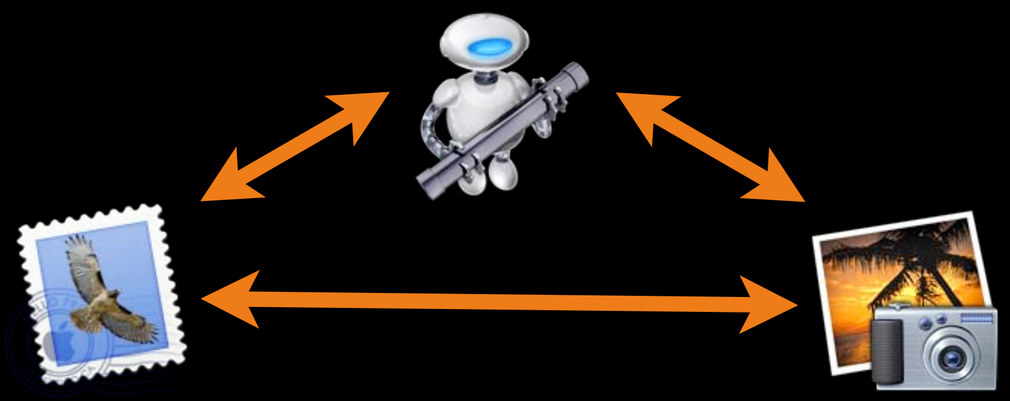


What's a Scripting Interface?

- A programmatic interface (an API)
- The objects and operations your application reveals to other programs
- Another view of your model
- An sdef (or something derived from one)
- Not just for AppleScript!

Why Have a Scripting Interface?

- All the usual reasons…
  - Automated workflows
  - Automated testing
  - Interoperability
  - Resistance to feature creep
  - Identification of user requirements
  - Make your application indispensable to your users
- Best way to tie your application into Automator!



Design Principles

Know your model

- Scriptability means model scripting, not view scripting
  - Easier to write
  - Easier to read
  - Model is more stable than view
  - Automator and AppleScript are based on model objects

Design Principles

Know your model

```
tell application “Address Book”
  select list view 2 of window 1
  type “Nebel”
  select menu item 15 of menu “Card”
end tell
```

```
tell application “Address Book”
  set my card to person “Chris Nebel”
end tell
```

Design Principles

Know your users

- The purpose of any object or command should be obvious to a user
  - Avoid internal implementation objects
- Helps plan your implementation
  - What to do first
  - What to do last!

Design Principles

Know your model

application *n* : The iTunes application program.

ELEMENTS
contains browser windows, encoders, EQ presets, EQ windows, playlist windows, sources, visuals, windows.

application *n* : The iPhoto application program.

ELEMENTS
contains albums, keywords, photos, windows.

Design Principles

Be object-oriented

- Organize your scripting interface around objects, not commands
  - Easier to understand
  - Gives a smaller, simpler interface
  - Exploits features of scripting languages (tell, whose, etc.)
- Only define new commands when you need to

Design Principles

Be object-oriented

CloseAllWindows *v* : Closes all windows

CloseAllWindows
→ boolean : Success

window *n* [inh. item]: A window.

RESPONDS TO
close.

Design Principles

Don’t re-invent

- Reuse standard definitions
- Use inheritance to make smaller object definitions
- Imitate similar applications
  - Text processing? Look at TextEdit
  - Personal data? Look at Address Book
  - Drawing application? Look at Sketch

Design Steps

- Read TN2106, Scripting Interface Guidelines
- Identify your model objects, their attributes, and their relationships
- Identify your unique operations
- Design your entire interface
  - Phase the implementation if necessary
  - It's ok to do less than your GUI
  - It's ok to do more than your GUI
- Put it all in an sdef
- Follow the guidelines!
  - Consistency is power

Hands On: What You'll See

- Add an sdef to your application
- Include the standard material
- Add two classes
  - One inherits from the other
  - Add a property
- Make an element relationship
- Add two commands
  - Object-first vs. verb-first dispatch
- Shortcuts!
  - desdp

Hands On

Writing an sdef for Sketch

http://developer.apple.com/wwdc2005/

Introduction to Sketch

- A simple drawing application
  - Draw boxes, circles, lines, and text
  - Complete source available in /Developer/Examples/AppKit

Open the Sketch Project

- Make a copy first
  - Go to /Developer/Examples/AppKit
  - Copy Sketch folder to where you want it
- Open the project

Add a Skeleton sdef

- Add the skeleton sdef file
  - Skeleton.sdef from sample code image (copy into project)
  - Rename it to Sketch.sdef
- Add Info.plist key
  - OSAScriptingDefinition = “Sketch.sdef”

Tweak Skeleton for Sketch

- Comment out “as” parameter for “save”
  - Sketch only has one document format; “as” is pointless
- Change “document” implementation class to “SKTDrawDocument”

Add Sketch Suite

- What's a suite?
  - Suites are organizational blocks to hold related elements
  - No impact on implementation; purely to help users find things
- After the Text suite…

```
<suite name="Sketch Suite" code="sktc"
       description="Sketch-specific classes and commands.">
</suite>
```

Design Sketch Scriptability

- Ideally, design scripting before writing code!
  - Helps design your code model also
- Failing that, analyze your existing model classes
  - A well-designed implementation model will align with scripting interface
- Do it…
  - Select the “Model Classes” group
  - Select Design > Class Model > Quick Model
  - Observe the class groupings
  - Observe the properties

Add “graphic” Class

- In the Sketch suite…

```
<class name="graphic" code="grph">
  <cocoa class="SKTGraphic"/>
  <property name="x position" code="xpos" type="real"/>
  <property name="y position" code="ypos" type="real"/>
  <property name="width" code="widt" type="real"/>
  <property name="height" code="heig" type="real"/>
  <property name="stroke color" code="sclr" type="color"/>
  <property name="fill color" code="fclr" type="color"/>
  <property name="stroke thickness" code="slwd" type="real">
     <cocoa key="strokeLineWidth"/>
  </property>
</class>
```

Add “graphic” Class

- Add element relationship (documents contain graphics)

```
<class name="document">
  <cocoa class="SKTDrawDocument"/>
  <element type="graphic"/>
</class>
```

Add “box” Class

- In the Sketch suite…

```
<class name="box" plural="boxes" code="d2rc" inherits="graphic"
  description="A rectangle graphic.">
  <cocoa class="SKTRectangle"/>
</class>
```

- Add element relationship (documents contain boxes)

```
<class name="document">
  <cocoa class="SKTDrawDocument"/>
  <element type="graphic"/>
  <element type="box"/>
     <cocoa key="rectangles"/>
  </element>
</class>
```

Add a Property

- Boxes have an orientation:

```
<class name="box" plural="boxes" code="d2rc" inherits="graphic">
  <cocoa class="SKTRectangle"/>
  <property name="orientation" code="orin" type="orientation"/>
</class>
```

- What's an orientation? An enumerated type:

```
<enumeration name="orientation" code="orie">
  <enumerator name="portrait" code="port"/>
  <enumerator name="landscape" code="land"/>
  <enumerator name="square" code="squa"/>
</enumeration>
```

Add a Command

Object-first dispatch

- Define the command:

```
<command name="rotate" code="sktcrota">
   <direct-parameter type="graphic"/>
   <parameter name="by" code="by  " type="real">
      <cocoa key="byDegrees"/>
   </parameter>
</command>
```

- Specify who understands it:

```
<class name="graphic">
   <responds-to name="rotate">
      <cocoa method="rotate:">
   </responds-to>
</class>
```

Add a Command

Verb-first dispatch

- Define the command:

```
<command name="align" code="sktcalig"
   description="Align a set of objects.">
   <cocoa class="SKTAlignCommand"/>
   <direct-parameter>
      <type type="graphic" list="yes"/>
   </direct-parameter>
   <parameter name="to" code="to  " type="edge">
      <cocoa key="toEdge"/>
   </parameter>
</command>
```

Add a Command

Verb-first dispatch

- Define the edge enumeration:

```
<enumeration name="edge" code="edge">
  <enumerator name="left edges" code="left"/>
  <enumerator name="right edges" code="righ"/>
  <enumerator name="horizontal centers" code="horc"/>
  <enumerator name="top edges" code="top "/>
  <enumerator name="bottom edges" code="bott"/>
  <enumerator name="vertical centers" code="verc"/>
</enumeration>
```

Run a Script!

```
tell application "Sketch"
  tell document 1
     set r to random number 65535
     set g to random number 65535
     set b to random number 65535
     set fill color of some graphic to {r, g, b}
  end tell
end tell
```

An Easier Way

- Use desdp(1) to translate an old-format scripting interface
  - Translates ‘aete’ resources and scriptSuite/scriptTerminology files
  - Not a complete solution; result will need editing

```
% desdp /build/Sketch.app | xmllint --format - > OldSketch.sdef
```

What You've Learned

- What a scripting interface is
- Why it's valuable
- Design guidelines for your interface
- How to write an sdef
  - Suites
  - Classes
  - Commands
- How to leverage your old scripting interface if you have one

More Information

http://developer.apple.com/wwdc2005/

Documentation, Sample Code, and Other Resources

http://developer.apple.com/wwdc2005/

Related Sessions

Implementing a Scripting Interface	Session 112 Wednesday 9:00AM
Lab: Make Your Application Scriptable	Wednesday 2:00PM

Who to Contact

Sal Soghoian
AppleScript and Automator Product Manager
sal@apple.com

Todd Fernandez
AppleScript and Automator Engineering Manager
trf@apple.com

Q&A

John Montbriand

http://developer.apple.com/wwdc2005/

WWDC2005

Worldwide Developers Conference